Лабораторная работа № 5
ОСНОВЫ ТЕХНОЛОГИИ СОКЕТОВ

Цель работы

Изучить:

- понятие сокета, типы сокетов TCP/IP;
- принципы реализации архитектуры клиент-сервер на основе интерфейса сокетов;
- основные классы и методы .NET Framework для разработки сетевых приложений с использованием Windows Sockets.

Постановка задачи

1.Изучить методические указания к лабораторной работе, материалы лекций и рекомендуемую литературу.

2.Разработать консольное клиент-серверное приложение, демонстрирующее взаимодействие на основе потоковых сокетов.

3.Разработать консольное клиент-серверное приложение, демонстрирующее взаимодействие на основе дейтаграммных сокетов.

4.Ответить на контрольные вопросы.

Методические указания

1. Понятие сокета

Сокет (Socket - гнездо, разъем) - абстрактное программное понятие, используемое для обозначения в прикладной программе конечной точки сетевого соединения.

Технология (интерфейс) сокетов – название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут выполняться как на одном компьютере, так и на разных, связанных между собой сетью.

Интерфейс сокетов был разработан в Калифорнийском университете в г. Беркли и встроен в ОС BSD Unix в 1983 г.

Важнейшим преимуществом сокетов является предоставление единого независимого интерфейса сетевого программирования для различных сетевых протоколов.

Библиотека Win32 Windows Sockets (Winsock) предоставляет механизмы программирования сокетов. В Microsoft .NET Framework имеется более высокий по отношению к Winsock уровень, благодаря чему управляемые приложения также могут взаимодействовать через сокеты (рис. 1).

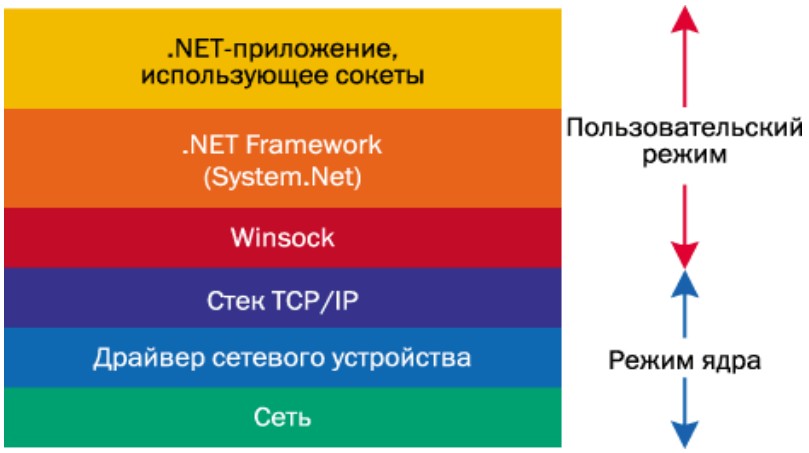

Рисунок 1 – Уровни сокетов Windows

Интерфейс сокетов расположен над транспортным уровнем стека TCP/IP, но в некоторых случаях может взаимодействовать напрямую с сетевым уровнем в обход транспортного.

2. Типы сокетов

Существуют три основных типа сокетов: потоковые, дейтаграммные и сырые.

Потоковые сокеты – это сокеты с установлением соединения, состоящие из потока байтов, который может быть двунаправленным. Т.е. через такую конечную точку приложение может и передавать, и получать данные. Потоковый сокет гарантирует обнаружение и исправление ошибок, обрабатывает доставку и сохраняет последовательность данных. Он подходит для передачи больших объемов данных, поскольку в этом случае накладные расходы, связанные с установлением соединения, незначительны по сравнению со временем передачи самого сообщения. Качество передачи достигается за счет использования протокола TCP.

Дейтаграммные сокеты – это сокеты без установления соединения, не обеспечивающие надежность при передаче. Применяются для приложений, когда неприемлемы затраты времени, связанные с установлением явного соединения. Для передачи данных используется протокол UDP.

Сырые сокеты (raw sockets - необрабатываемые, простые) – это сокеты, которые взаимодействуют с протоколами сетевого уровня в обход протоколов транспортного уровня. Используются для непосредственного доступа приложения к IP-пакетам сетевого уровня. Использование сырых сокетов возможно при разработке низкоуровневого системного ПО. Например, сырые сокеты используют различные программы-анализаторы пакетов, сниферы, утилиты TCP/IP ping, tracert и т д.

3. Адресация сокетов. Номера портов

Интерфейс сокетов поддерживает разнообразные сетевые стеки протоколов: TCP/IP, IPX/SPX, NetBIOS/SMB, AppleTalk, ATM, Infrared Sockets. Каждому из них соответствует свое семейство адресов сокетов. Напрмер, стеку TCP/IP соответствует семейство адресов InterNetwork для адресов IPv4, InterNetworkV6 для адресов IPv6. Стеку IPX/SPX соответствует семейство адресов Ipx и т.д.

Семейство адресов – важнейший параметр сокета. Он указывает используемый в настоящее время сетевой протокол и ограничивает применение других параметров сокета.

Мы рассмотрим адресацию сокетов только для стека протоколов TCP/IP, как самого распространенного на сегодняшний день.

Адрес сокета при использовании протоколов TCP/IP – это IP-адрес и номер порта прикладной службы.

IP-адрес уникален в Internet. Номер порта уникален на отдельном компьютере. Следовательно, адрес сокета будет уникален в Internet. Это позволяет удаленным процессам общаться исключительно на основе адреса сокета.

При работе с сокетами могут использоваться некоторые специальные IP-адреса. Например, для отладки клиент-серверного приложения на одном компьютере применяется адрес петли обратной связи 127.0.0.1 или соответствующее ему имя localhost.

Для получения информации на всех сетевых интерфейсах локального узла используется специальный адрес 0.0.0.0.

Порт задается, чтобы решить задачу одновременного сетевого взаимодействия с несколькими приложениями на одном узле. Если на компьютере выполняется несколько приложений, то получая пакет из сети, можно идентифицировать приложение по уникальному номеру порта, который задан при установлении связи.

Программисты должны быть внимательны при выборе номера порта, поскольку некоторые доступные порты зарезервированы для использования популярными службами, такими как FTP, HTTP и т.д. Эти порты обслуживаются и распределяются центром Internet Assigned Numbers Authority (IANA), их описание содержится в RFC 1700.

Номера портов разделяются на 3 категории (стандартные, зарегистрированные и динамические и/или частные):

- Номера от 0 до 1023 зарезервированы для стандартных служб.
- Порты с номерами от 1024 до 49151 являются регистрируемыми.
- Порты с номерами от 49152 до 65535 - динамические и частные порты.

Во избежание накладок с портами, уже занятыми системой или другим приложением, ваша программа должна выбирать порты, начиная с 1024. Можно вместо конкретного номера порта задать 0, тогда система сама выберет произвольный неиспользуемый в данный момент номер.

Запустив утилиту netstat –a, можно увидеть перечень всех используемых в данный момент на компьютере номеров портов.

В файле services из каталога <windir>\system32\drivers\ets перечислены предопределенные пользовательские и системные номера портов, используемых стандартными службами. Если порт содержится в перечне этого файла, то утилита netstat вместо номера порта отобразит имя протокола.

4. Порядок байт

В памяти компьютера IP-адрес и номер порта представляются в ***системном порядке (host-byte-order)***. Для Intel-совместимых процессоров это порядок от менее значимого к более значимому байту. Его называют ***обратный*** или ***little endian*** (младший байт числа расположен в младшем адресе памяти). Для других типов процессоров, например Motorola, используется ***прямой порядок*** или ***big endian*** (в младшем адресе сохраняется старший байт числа).

Сетевые стандарты требуют, чтобы многобайтные значения передавались в ***сетевом порядке (network-byte order)***. Это прямой порядок следования байтов. Поэтому в некоторых случаях, когда системный и сетевой порядок байт не совпадает, существует необходимость преобразования чисел из одной формы в другую. Для этого разработаны специальные функции и методы.

5. Коммуникационная модель клиент-сервер

Приложение, использующее сокеты, состоит из распределенной программы, исполняемой на обоих концах канала связи. Программу, инициирующую передачу, называют ***клиентом***. ***Сервер*** представляет собой модуль, пассивно ожидающий входящих запросов на установку соединений от удаленных клиентов. Как правило, серверное приложение загружается при запуске системы и прослушивает свой порт, ожидая входящих соединений. Клиентские приложения пытаются установить соединение с сервером, после чего начинается обмен данными. По завершении сеанса связи клиент, как правило, разрывает соединение. На рисунке представлена базовая модель взаимодействия для потоковых сокетов.

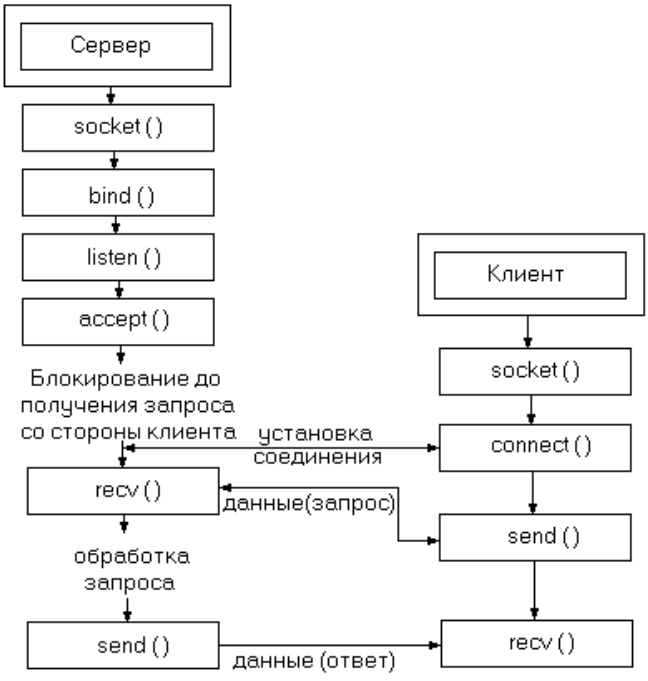

Рисунок 2 - Блок-схема взаимодействия потоковых сокетов

6. Классы для работы с адресами в .NET

Пространство имен System.Net содержит классы для работы с доменными именами и IP-адресами: Dns, IPHostEntry, IPAddress, IPEndPoint и др..

Чтобы получить IP-адрес из DNS-имени хоста можно использовать статический метод Resolve класса Dns. У одного хоста может быть несколько IP-адресов и альтернативных имен. При получении IP-адреса метод Resolve возвращает объект класса IPHostEntry, который содержит массив адресов, имя хоста и альтернативные имена.

Например, получим адресную информацию для хоста www.microsoft.com:

```
string hostname = “www.microsoft.com”;
IPHostEntry host = Dns.Resolve(hostname);
```

В классе Dns есть различные статические методы, возвращающие объекты IPHostEntry. Они приведены в таблице 1.

Таблица 1 – методы класса Dns

Метод Dns	Описание
Resolve()	По Dns-имени или IP-адресу в десятично-точечной нотации получает IPv4-адреса и альтернативные имена хоста
GetHostEntry()	По Dns-имени или IP-адресу в десятично-точечной нотации получает IPv6-адреса и альтернативные имена хоста
GetHostByName()	По заданному Dns-имени хоста возвращает список IP-адресов
GetHostByAddress()	Получает строку IP-адреса в десятично-точесной нотации или объект IPAddress, возвращает объект IPHostEntry
GetHostName()	Возвращает имя локального хоста

IP-адреса обрабатываются в классе IPAddress. Используя свойство AddressList выберем из полученного с помощью класса Dns списка IP-адресов хоста первый:

```
IPAddress ipAddr = host.AddressList[0];
```

Используя класс IPEndPoint, создадим локальную конечную точку для сервера. Конечная точка задает адрес сокета – это комбинация IP-адреса и номера порта. Выбираем свободный номер порта из соответствующего диапазона:

```
int i = 11000;
IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, i);
```

Для создания объекта IPAddress можно использовать статический метод Parse(). Например:

```
IPAddress address = IPAddress.Parse(“172.16.56.2”);
IPAddress address = IPAddress.Parse(defaultGateway);
```

Этот метод преобразует строку с IP-адресом в десятично-точечной нотации в целое 4-хбайтовое число с сетевым порядком следования байт. К полю с IP-адресом можно обратиться с помощью свойства IPAddress.Address.

В классе IPAddress есть несколько доступных только для чтения полей, которые возвращают специальные IP-адреса:

- IPAddress.Loopback возвращает адрес петли обратной связи 127.0.0.1;
- IPAddress.Broadcast возвращает широковещательный адрес 255.255.255.255;
- IPAddress.Any возвращает адрес 0.0.0.0, который можно использовать для получения сообщений на всех сетевых интерфейсах данного узла.

Класс IPAddress также содержит методы для преобразования порядка следования байт: IPAddress.NetworkToHostOrder(), IPAddress.HostToNetworkOrder().

7. Поддержка сокетов в .NET

Поддержку сокетов в .NET обеспечивают классы в пространстве имен System.Net.Sockets. Основные классы приведены в таблице 2.

Таблица 2 - Классы в пространстве имен System.Net.Sockets для работы с сокетами

Класс	Описание
NetworkStream	Реализует базовый класс потока, из которого данные отправляются и в котором они получаются. Это абстракция высокого уровня, представляющая соединение с каналом связи TCP/IP.
Socket	Класс, обеспечивающий базовую функциональность приложения на основе сокетов.
TcpClient	Данный класс строится на классе Socket, чтобы обеспечить TCP-обслуживание на более высоком уровне. TcpClient предоставляет несколько методов для отправки и получения данных через сеть.
TcpListener	Построен на низкоуровневом классе Socket. Предназначен для создания серверных приложений. Он ожидает входящие запросы на соединения от клиентов и уведомляет приложение о любых соединениях.
UdpClient	Предназначен для реализации обслуживания по протоколу UDP.
SocketException	Это исключение порождается, когда в сокете возникает ошибка.

8. Класс Socket.

Класс Socket является базовым. Методы этого класса выполняют различные проверки, связанные с безопасностью, а затем переправляются к соответствующим аналогам из Windows Sockets API.

Важнейшие свойства класса Socket описаны в таблице 3.

Таблица 3 - Свойства класса Socket

Свойство	Описание
AddressFamily	Задает семейство адресов сокета – значение из перечисления SocketAddressFamily.
Available	Возвращает объем доступных для чтения данных.
Blocking	Дает или устанавливает значение, показывающее, находится ли сокет в блокирующем режиме.
Connected	Возвращает значение, информирующее, соединен ли сокет.
LocalEndPoint	Дает локальную конечную точку.
ProtocolType	Дает тип протокола сокета.
RemoteEndPoint	Дает удаленную конечную точку сокета.
SocketType	Дает тип сокета.

Основные методы класса Socket представлены в таблице 4.

Таблица 4 – Методы класса Socket

Метод	Описание
Accept()	Создает новый сокет для обработки входящего запроса на соединение.
Bind()	Связывает сокет с локальной конечной точкой для ожидания входящих запросов на соединение.
Close()	Закрывает сокет.
Connect()	Устанавливает соединение с удаленным хостом.
GetSocketOption()	Возвращает значение SocketOption.
IOControl()	Устанавливает для сокета низкоуровневые режимы работы. Этот метод обеспечивает низкоуровневый доступ к лежащему в основе экземпляру класса Socket.
Listen()	Помещает сокет в режим прослушивания. Этот метод предназначен только для серверных приложений.
Receive()	Получает данные от соединенного сокета.
ReceiveFrom()	Принимает дейтаграмму в буфер данных и сохраняет конечную точку.
Poll()	Определяет статус сокета.
Select()	Проверяет статус одного или нескольких сокетов.
Send()	Отправляет данные соединенному сокету.
SendTo()	Отправляет дейтаграмму на указанную конечную точку.
SetSocketOption()	Устанавливает опцию сокета.
Shutdown()	Запрещает операции отправки и получения данных на сокете.

Чтобы настроить серверный TCP-сокет в управляемом коде, прежде всего нужно создать экземпляр класса Socket. Его конструктор принимает три параметра: AddressFamily, SocketType и ProtocolType. Параметр AddressFamily определяет используемую сокетом схему адресации. Чаще всего в качестве этого параметра используются значения InterNetwork (для адресов IPv4) и InterNetworkV6 (для адресов IPv6). Параметр SocketType определяет тип сокета. Например, Stream для потоковых сокетов, и Dgram для дейтаграммных сокетов. Параметр ProtocolType определяет применяемый сокетом протокол и принимает такие значения, как Tcp, Udp, Idp, Ggp и т. д.

Например, создать сокет для взаимодействия по протоколу TCP можно следующим образом:

```
Socket s = new Socket(IPEndPoint.Address.AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
```

Как только потоковый сокет на сервере создан, его необходимо привязать к адресу. Привязка клиентского сокета к адресу не обязательна. Чтобы привязать сокет к адресу, используется метод Bind объекта Socket. Этому методу нужен адрес и порт, которые будут сопоставлены с сокетом, поэтому в качестве параметра он принимает экземпляр класса, производного от EndPoint. Как правило, это объект класса IPEndPoint (кроме него в .NET Framework входит только один класс, производный от EndPoint, — IrDAEndPoint, который служит для взаимодействия посредством инфракрасного порта).

После вызова метода Bind метод Socket.Listen переводит сокет в режим прослушивания и конфигурирует для сокета внутренние очереди. Когда клиент пытается подключиться к серверу, в очередь помещается запрос на установление соединения. Метод Listen принимает один аргумент — максимальное число запросов соединений, которые могут находиться в очереди.

Метод Socket.Accept извлекает из очереди первый запрос, устанавливает соединение с клиентом и возвращает новый объект Socket, который можно использовать для коммуникационного взаимодействия с данным клиентом.

В клиентской части приложения после создания объекта Socket обычно вызывается метод Connect. Можно также сначала вызвать метод Bind, если нужно, чтобы клиент использовал конкретный порт для связи с сервером. Если вы не связали клиентский сокет с портом, метод Connect выберет порт клиента автоматически. При вызове Connect клиентское приложение пытается установить соединение с сервером. Метод Connect также принимает объект EndPoint, через который определяется целевой удаленный хост. Как только соединение установлено, клиент и сервер могут передавать данные методами Send и Receive.

Когда приложения завершат обмен данными, необходимо закрыть сокет. Чтобы корректно инициировать закрытие сокета, вызывается метод Shutdown. Это позволяет передать все неотправленные данные и получить еще не принятые данные из буфера. Если вы синхронно читаете данные из сокета и получаете 0 байтов, значит, вторая сторона закрыла сокет.

Если при выполнении какого-либо метода сокета возникает сетевая ошибка, генерируется исключение SocketException. Этот объект является оболочкой кода ошибки, полученного от Win32. SocketException.ErrorCode — тот же код ошибки, который вы получили бы, вызвав Winsock-функцию WSAGetLastError. Основные коды ошибок Winsock приведены в таблице 5.

Таблица 5 – Основные коды ошибок

Номер ошибки	**Значение Winsock**	**Значение SocketError**	**Описание**
10004	WSAEINTR	Interrupted	Системный вызов прерван. Это может произойти, если выполняется вызов сокета, асокет закрыт.
10048	WSAEADDRINUSE	AddressAlreadyInUse	Адрес, к которому вы пытаетесь привязать сокет или который вы хотите прослушивать, уже занят.
10053	WSACONNABORTED	ConnectionAborted	Соединение было прервано локальным компьютером.
10054	WSAECONNRESET	ConnectionReset	Соединение было прервано удаленным компьютером.
10061	WSAECONNREFUSED	ConnectionRefused	Соединение с конечной точкой отклонено. Это может произойти, если хост отключен или если порт занят, а очередь запросов заполнена.

9. Дейтаграммные сокеты.

Создать детаграммный сокет можно следующим образом:

Socket s = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);

При использовании дейтаграммных сокетов сначала создают сокет. Затем выполняют привязку сокета к интерфейсу, на котором будут принимать данные, методом Bind (как и в случае протоколов, ориентированных на соединения). Разница в том, что вместо использования методов Listen или Accept нужно просто ожидать приема входящих данных. Поскольку в этом случае соединения нет, принимающий сокет может получать дейтаграммы от любой машины в сети.

Простейший метод приема — ReceiveFrom:

Другой способ приема (отправки) данных на сокетах, не требующих соединения, — установление соединения (хоть это и звучит странно). После создания сокета можно вызвать метод Connect. Фактически никакого соединения не происходит. Адрес сокета, переданный в функцию соединения, ассоциируется с сокетом, чтобы было можно использовать метод Receive вместо ReceiveFrom (поскольку источник данных известен).

Есть два способа отправки данных через сокет, не требующий соединения. Первый и самый простой — создать сокет и вызвать метод SendTo.

Как и при получении данных, сокет, не требующий соединения, можно подключать к адресу конечной точки и отправлять данные методом Send. После создания этой привязки использовать для обмена данными метод SendTo с другим адресом нельзя — будет выдана ошибка. Отменить привязку сокета можно, лишь вызвав метод Close с описателем этого сокета, после чего следует создать новый сокет.

Поскольку соединение не устанавливается, его формального разрыва или корректного закрытия не требуется. После прекращения отправки или получения данных отправителем или получателем просто вызывается метод Close с описателем требуемого сокета, в результате чего освобождаются все выделенные ему ресурсы.

Задания на лабораторную работу

Основные задания (обязательные для выполнения)

Задание 1.

Используя класс Socket пространства имен System.Net.Sockets .Net Framework, разработать синхронное консольное клиент-серверное приложение. Клиент и сервер должны осуществлять взаимодействие по протоколу TCP.

В серверной части вывести на экран дескриптор, IP-адрес и порт слушающего сокета. При получении от клиента запроса на установление соединения вывести на экран новый дескриптор сокета, а также IP-адрес и номер порта подключившегося клиентского сокета.

Клиент и сервер должны запросить и получить друг от друга текущие дату и время.

Выводить на экран все отправленные и полученные по сокету данные в клиентской и северной части приложения.

Выполнять проверку и обработку ошибок.

В отчете привести блок-схему, листинг клиентской и серверной части приложения, а также копии экранов.

Задание 2.

Разработать клиент-серверное приложение, аналогичное предыдущему заданию, но для взаимодействия клиента и севера использовать протокол UDP и дейтаграммные сокеты.

В отчете привести блок-схему, листинг клиентской и серверной части приложения, а также копии экранов.

Дополнительные задания по вариантам

Вариант 1

Разработать клиент-серверное приложение для получения списка открытых TCP-портов удаленного хоста. Взаимодействие клиента и сервера осуществлять на основе потоковых сокетов.

Для получения портов использовать класс SocketException. Сканирование портов выполнять в цикле по номерам портов.

Вариант 2

Разработать клиент-серверное приложение для получения списка открытых UDP-портов удаленного хоста. Взаимодействие клиента и сервера осуществлять на основе потоковых сокетов.

Для получения портов использовать класс SocketException. Сканирование портов выполнять в цикле по номерам портов.

Вариант 3

Разработать клиент-серверное приложение для получения списка IPv4-адресов и масок сетевых интерфейсов удаленного узла. Взаимодействие клиента и сервера осуществлять на основе потоковых сокетов. Для получения адресов использовать пространство имен System.Net.NetworkInformation, классы NetworkInterface, IPInterfaceProperties (свойство UnicastAddresses).

Вариант 4

Разработать клиент-серверное приложение для получения списка IPv6-адресов сетевых интерфейсов удаленного узла. Взаимодействие клиента и сервера осуществлять на основе потоковых сокетов. Для получения адресов использовать классы NetworkInterface, IPInterfaceProperties (свойство UnicastAddresses).

Вариант 5

Разработать клиент-серверное приложение для получения DNS-имени удаленного узла (имя узла + имя домена), а также IP-адреса DNS-сервера удаленного узла. Взаимодействие клиента и сервера осуществлять на основе потоковых сокетов.

Вариант 6

Разработать клиент-серверное приложение для получения списка физических адресов сетевых интерфейсов удаленного узла. Взаимодействие клиента и сервера осуществлять на основе потоковых сокетов. Для получения адресов использовать классы пространства имен System.Net.NetworkInformation (класс PhysicalAddress и др).

Вариант 7

Разработать клиент-серверное приложение для получения списка активных TCP-портов удаленного хоста. Взаимодействие клиента и сервера осуществлять на основе потоковых сокетов.

Для получения портов использовать класс IPGlobalProperties.

Варианты 8-14

Задания аналогичные вариантам 1-7, только с использованием дейтаграммных сокетов для взаимодействия клиента и сервера.

Контрольные вопросы

1. Что такое сокет? Как по отношению к уровням стека TCP/IP расположен интерфейс сокетов?
2. Сколько сокетов необходимо для взаимодействия клиента и сервера? Что представляет собой адрес сокета?
3. Назовите типы сокетов. В каком случае предпочтительнее использовать тот или иной тип сокетов? Какой тип сокетов обеспечивает надежность и порядок доставки?
4. Чем различаются потоковые протоколы и протоколы, ориентированные на передачу сообщений? Как это отражается на коде при написании программ?
5. Чем отличается процесс получения и отправки данных на сокете, не требующем соединения?
6. Какой адрес можно использовать, чтобы прослушивать все сетевые интерфейсы локального узла?
7. Какой порядок байтов используется в Intel-совместимых процессорах? Какой порядок байтов применяется для работы с сокетами?
8. Какие номера портов могут задействовать прикладные программы при работе с сокетами?
9. Как осуществляется корректное завершение сеанса работы с потоковым сокетом, с дейтаграммным сокетом?
10. Как можно узнать номера портов, занятых стандартными службами? В каком файле хранится эта информация?

11. Какие пространства имен и классы в .NET Framework обеспечивают поддержку сокетов?

Литература

1. Кумар В., Кровчик Э и др. .NET. Сетевое программирование / Пер. с англ. –М.: «Лори», 2007
2. Библиотека MSDN